

Projet P3 - 3ème année DLM

Orientation : Développement logiciel et multimédia (DLM)

Camelote

Fait par :

Bruno Alexandre Da Cruz Costa

Sous le direction de :

Prof. Julien Senn

Haute École Arc, HES-SO

Remerciements

Je souhaite remercier, M. Julien Senn et M. Benoît Le Callennec pour leur aide tout au long du projet.

Résumé

Cette documentation traite du parcours emprunté pour le développement d'un jeu créé sur Unity utilisant l'APK de Google Maps. Chaque étape importante sera soulignée et détaillée afin de comprendre l'entièreté du projet.

Camelote est un jeu semblable au légendaire Pokémon Go où l'utilisateur devra se déplacer dans la vraie vie pour faire bouger son personnage dans le jeu. Le but est de récupérer des ressources disponibles dans des points d'intérêts parsemés sur la carte et les utiliser pour améliorer son château et ses unités pour finalement devenir le plus grand Roi de Camelote.

L'application est dans un état stable et respecte en grande partie le cahier des charges. Le joueur peut se déplacer, interagir avec les points d'intérêts en récupérer les ressources et améliorer son château. Elle est également prête à accueillir les améliorations prévues des objectifs secondaires et tertiaires. Le programme est donc fonctionnel et les potentiels ajustements seront détaillés en conclusion de ce rapport.

Abstract

This documentation deals with the path taken for the development of a game created on Unity using the Google Maps APK. Each important step will be highlighted and detailed in order to understand the whole project.

Camelote is a game similar to the legendary Pokémon Go where the user will have to move in real life to make his character move in the game. The goal is to collect resources available in points of interest dotted around the map and use them to improve his castle and units to eventually become the greatest King of Camelot.

The application is in a stable state and largely meets the specifications. The player can move around, interact with points of interest, collect resources and upgrade his castle. It is also ready for the planned improvements of secondary and tertiary objectives. The program is therefore functional and the potential adjustments will be detailed in the conclusion of this report.

Table des matières

Remerciements	i
Résumé	iii
Abstract	v
1 Introduction	1
1.1 Mise en contexte	1
1.2 Le sujet du projet	1
1.3 Les détails du jeu	1
1.4 Cahier des charges	1
2 Analyse	3
2.1 Idée de base	3
2.2 Récupération du projet initial	4
2.3 Besoins du projet - Maquettes	5
2.4 Solution retenue	7
2.5 Planification	7
3 Conception	9
3.1 Besoins de l'application	9
3.2 Base de données	11
3.3 Structure de l'application	12
3.4 Langages, outils et frameworks utilisés	13
4 Réalisation	15
4.1 Managers	15
4.2 Base de données	17
4.3 Interface graphique	20
4.4 Points d'intérêts et ressources	24
4.5 Château	25
4.6 Tests	26
5 Résultats	29
5.1 Objectifs atteints	29
5.2 Objectifs non-atteints	29
5.3 État actuel de l'application	30
6 Conclusion	35
Annexes	V

Chapitre 1

Introduction

Ce rapport contient chaque étape et procédé, de l'analyse à l'implémentation, du projet P3 "Camelote"

1.1 Mise en contexte

Ce projet a été réalisé dans le cadre du module de troisième année, le P3, durant lequel, l'application Camelote a été construite sur la plateforme Unity en langage C#, une base de données a été définie pour contenir les données du jeu et assurer une persistance. La communication entre la base de données et l'application Unity a été réalisée par des scripts PHP utilisés en tant que Web Services. La particularité de ce projet est qu'il est basé sur celui d'un ancien étudiant. En effet, une partie sera récupérée d'une application fonctionnelle et existante qui gère la géolocalisation ainsi que les déplacements en direct de l'utilisateur.

1.2 Le sujet du projet

Comme pour tous les P3, l'étudiant a dû choisir son projet sur le site web "ape.ing.he-arc.ch". Le projet sélectionné est "Jeu Unity - Google Maps" où l'objectif est de créer un jeu de A à Z qui sera basé sur l'APK de Google Maps faite pour Unity et jouable sur un téléphone android. L'import des fonctionnalités de Google Maps, comme la localisation, fera donc partie du noyau du gameplay du jeu. En partant de cette base, il a été décidé de créer un jeu reprenant les principes de la célèbre application Pokémon Go. C'est ainsi que Camelote, un jeu reprenant le mythe Arthurien, est né.

1.3 Les détails du jeu

Le but du jeu est simple. En premier lieu, l'utilisateur apparaît sur une représentation 3D d'une carte de son environnement, si le joueur se trouve à Neuchâtel, alors la carte du jeu sera celle de la ville Suisse. Il sera alors représenté par un cylindre rouge qui indique également sa position. À partir de là, l'utilisateur pourra accéder à différents menus ainsi qu'à des points d'intérêts pour récupérer des ressources et recruter des unités.

Ces unités seront utilisables dans des combats d'arènes permettant de gagner encore plus de ressources. Les ressources forment le point centrale du jeu, avec celles-ci, l'utilisateur pourra créer et améliorer son château et renforcer ses recrues pour devenir le roi le plus puissant de Camelote.

1.4 Cahier des charges

En tant que propre client, le cahier des charges a été rédigé par l'étudiant ainsi que signé et validé par l'enseignant responsable du projet. La version pdf de ce cahier des charges est disponible en annexe de ce rapport [1].

1.4.1 Objectifs du cahier des charges

L'objectif principal du cahier des charges est d'apprendre à créer un jeu de bout en bout tout en apprenant les bases importantes du moteur de rendu Unity.

Quant aux objectifs du projet, ils sont mis en formes dans le cahier des charges mis en annexe.

Chapitre 2

Analyse

L'analyse de ce projet est séparée en trois parties :

1. La première est la création de l'idée du jeu. En effet, la création d'un jeu de A à Z fait partie des objectifs du projet. Ainsi, il a fallu trouver, inventer et mettre en place une idée utilisant des notions de localisation.
2. La seconde est l'analyse et la compréhension du projet provenant de l'étudiant de 3ème année précédemment énoncé.
3. La dernière est l'analyse des besoins du projet final, quelles sont les fonctionnalités nécessaires ou encore quel sera le comportement de l'utilisateur lorsqu'il voudra jouer au jeu.

2.1 Idée de base

De nos jours, il existe une pléthore de jeu utilisant la géolocalisation, nous pouvons énumérer Pokemon GO, Minecraft Earth ou encore The Walking Dead. Le plan était donc de s'aider de l'état de l'art afin de créer un système de gameplay inspiré ou presque similaire.

Ainsi, l'idée du jeu Camelote a été créée, un jeu où nous jouons le jeune roi Arthur dans sa quête d'une Camelote grande et unifiée, son but est donc de récupérer suffisamment de ressources et de soldats pour établir son règne.

2.1.1 Géolocalisation, points d'intérêts et ressources

Comme dans la plupart des jeux énoncés ci-dessus, Camelote utilise un système de points d'intérêt, similaires aux célèbres "PokéStops", disposés çà et là sur la carte du monde. L'utilisateur pourra alors interagir avec eux pour accéder à un menu permettant de récupérer des ressources ou encore recruter des unités. Là où un "PokéStop" donnera des "Pokéballs", les points d'intérêt de Camelote donneront du bois, de la nourriture pour les ouvriers ou encore du fer.

Ces ressources sont utilisées pour de multiples objectifs et sont le nerf de la guerre au sein de Camelote. C'est avec celles-ci que le joueur pourra renforcer son château, améliorer son arsenal ou encore entraîner ses unités. C'est pourquoi la récupération de ressources via l'exploration de points d'intérêts est essentielle au gameplay du jeu.

2.1.2 Arènes

Certains points d'intérêts sur la carte seront ce qui s'appelle des "Arènes". C'est dans celle-ci que l'utilisateur pourra envoyer ses unités au front pour capturer le terrain pour amasser un pactole de ressources. La bataille fonctionne sur un système passif où l'utilisateur déposera simplement ses unités et le résultat du combat changera selon leurs puissances.

2.1.3 Unités

Les unités recrutables sont séparées en trois types :

- Épéiste

- Lancier
- Archer

Elles fonctionnent via un système de "ChiFuMi", l'épéiste bat l'archer, l'archer bat le lancier et le lancier bat l'épéiste. Cependant, bien que ce triangle existe, une unité très bonne à l'épée pourra battre une unité médiocre à la lance. En effet, chaque unité possède trois caractéristiques influant sa manière de se battre. La première est la force, qui permet d'augmenter la force de ses coups, la seconde est la résistance qui du coup permet de subir moins de dégâts et finalement le potentiel, qui est une caractéristique fixe qui détermine la montée des deux autres à chaque montée de niveau.

2.1.4 Château

Enfin viens le concept du château dont l'utilisateur a accès. Ce château peut être amélioré avec des ressources et chacun de ces progrès offre plusieurs fonctionnalités : une caserne permettant d'accueillir plus d'unités et des réservoirs de ressources plus grands. De plus, son apparence change également, se transformant en une véritable forteresse une fois le niveau maximum atteint.

2.1.5 Attente temps réel

La quasi-totalité des jeux mobiles actuels utilise ce qu'on appelle l'attente temps réel. En effet, lors d'une action de grande envergure, le but de l'attente temps réel est d'afficher un minuteur demandant à l'utilisateur d'attendre que la tâche se termine. Dans le cadre de Camelote, le but est d'ajouter cet aspect d'attente réel sur quelques fonctionnalités :

- L'amélioration du château
- Le combat dans une arène
- La récupération de ressources dans un point d'intérêt

2.2 Récupération du projet initial

Camelote se base sur une application existante créée lors d'un projet P3 d'un ancien étudiant de l'école HE-Arc. Ce projet, ainsi que sa documentation, ont été analysés en détail pour bien comprendre sur quelles bases le jeu allait être construit et établi.

L'application créée par l'étudiant, "Guided Tour Neuchatel", est un jeu simple où l'utilisateur doit se déplacer dans une carte réalisée via Google Maps avec un joystick tactile et récupérer de petits éléments générés aléatoirement afin d'augmenter le score final du joueur. Le but étant d'accumuler le plus de points pour arriver dans le tableau des High-Scores. Nonobstant la grande aide qu'a été le projet pour l'implémentation de la carte, pas tous ces éléments étaient judicieux au projet Camelote. En effet, après l'analyse du projet Unity, plusieurs éléments inutiles ou ne convenant pas aux besoins du projet P3 actuel ont été retirés. Si bien qu'au final, la décision a été de créer un projet vierge et de récupérer uniquement l'essentiel.

2.2.1 Structure de la Scène Unity

Dans la scène Unity, la partie "Game" et le "manager" communiquant avec la base de données ont été récupérés. Cette partie contient le personnage, la caméra et la carte ainsi que sa gestion Google Maps. Le reste a été jugé inutile car pas à jour. L'interface graphique ne convenait pas.

2.2.2 Scripts

Deux scripts C# ont été récupérés du premier projet :

- Le premier est "PlayerMovementsLocations" qui gère la génération de la carte ainsi que les déplacements en direct du joueur.
- Le second est la récupération et génération des points d'intérêts du jeu, le script "LoadPointOfInterest"

2.3 Besoins du projet - Maquettes

Pour définir en détail comment l'application doit être, quelques maquettes représentant les différents menus et interfaces ont été conçues. Ces maquettes sont la première étape de définition du projet, c'est par celles-ci que les premiers besoins de l'application seront détaillés.

2.3.1 Menu de lancement du jeu

C'est dans ce menu que l'utilisateur pourra lancer le jeu.

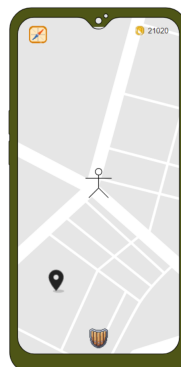
FIGURE 2.1 – Maquette d'un menu principal



2.3.2 Jeu

Le jeu montre l'utilisateur au centre de l'écran, les différents points d'intérêts aux alentours et le bouton principal permettant d'ouvrir le menu.

FIGURE 2.2 – Maquette de l'interface du jeu



2.3.3 Menu du jeu

Le menu du jeu donne accès aux autres menus via des boutons. Le bouton du château, des unités et des ressources sont représentés via des icônes simple.

FIGURE 2.3 – Maquette du menu de jeu



2.3.4 Menu des ressources

Ce menu indique la quantité de ressources que l'utilisateur possède.

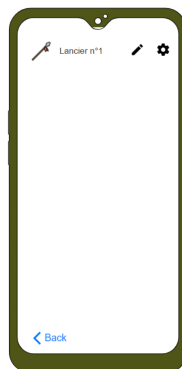
FIGURE 2.4 – Maquette du menu des ressources



2.3.5 Menu des unités

Le menu des unités montre le type, le nom et les menus de toutes les recrues de l'utilisateur

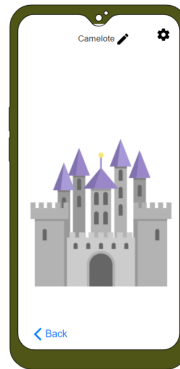
FIGURE 2.5 – Maquette du menu des unités



2.3.6 Gestion du château

C'est ici que l'utilisateur pourra visualiser l'état de son château ainsi que de l'améliorer

FIGURE 2.6 – Maquette du menu du château



2.4 Solution retenue

À la fin de l'analyse, le jeu prenait cette forme :

- L'utilisateur lance le jeu
- Se connecte
- Arrive sur la carte représentant ses alentours
- Il a accès à des points d'intérêts
- Peut se rapprocher en se déplaçant dans la vraie vie
- Récolter des ressources et/ou recruter des unités
- Il peut ensuite améliorer ses nouvelles recrues
- Améliorer son château
- Prendre d'assaut les arènes de sa ville

2.5 Planification

C'est dans la phase de planification que les différents éléments étudiés plus haut sont séparés en tâches et mis en forme dans un échéancier comme un diagramme de GANTT ou un diagramme de PERT. Elles sont ordonnées par ordre de priorité et une valeur estimant le temps correspondant au travail de la tâche y est indiquée.

2.5.1 GANTT et GitLab

GANTT

Le diagramme de GANTT a été créé et confirmé par M. Senn assez tôt dans le projet. Il permet d'organiser et d'ordonner les tâches du projet et ainsi indique, ou du moins estime, la quantité de travail pour amener le projet à son terme. De plus, il est possible d'y entrer un suivi des heures passées sur les différentes tâches du projet. Le diagramme de GANTT est disponible en entier en annexe [2].

GitLab

Sur la forge de l'école nommée GitLab, les *issues* et les *boards* ont été utilisés afin de lister les tâches terminées et celles qui sont encore à effectuer. Le wiki disponible a également été abondamment rempli de pages importantes comme le cahier des charges, un résumé de la conception ou encore le journal de travail.

2.5.2 Retard

Au cours du projet, un retard s'est généré à cause de quelques facteurs importants.

- Le premier est l'utilisation d'une technologie inconnue à l'étudiant. UI ToolKit est un package encore expérimental de Unity qui permet de créer des interfaces graphiques jolies de manière

simple. Cependant il est peu documenté et le temps attribué à son utilisation et sa découverte ont été mal estimé

- Le second est simplement le surplus de travail provenant des autres branches. Des projets ayant un rendu plus tôt dans l'année et qui nécessitait un travail prioritaire ont légèrement empêché le progrès dans l'implémentation du projet

Ce retard a été géré par une augmentation du rythme de travail sur la fin du projet. De par ce fait, certaines tâches n'ont pas été réalisées à temps.

2.5.3 Amélioration de la gestion de planning

Une fois le projet terminé, le planning a été analysé minutieusement. Plusieurs facteurs permettant d'améliorer une future gestion de projet ont été trouvés :

- Une recherche plus approfondie sur les tâches avant d'en estimer le temps nécessaire
- Ne pas oublier de noter une tâche et de perdre du temps en milieu de projet

Les objectifs secondaires de l'application n'ont pas pu être atteints et sont de bonnes voies pour améliorer le projet.

Chapitre 3

Conception

La conception est la dernière étape du travail avant de commencer l'implémentation et la réalisation du projet. Via celle-ci, les concepts et résultats de l'analyse précédemment faite sont mises en formes par des schémas ou des diagrammes.

C'est également ici que les premiers détails liés au code, par exemple quel langage/framework ou encore les bibliothèques employées, sont définis.

3.1 Besoins de l'application

Camelote, et comme tous les projets, doit passer par une étude de besoins pour définir ses caractéristiques et ses fonctionnalités. Ces besoins sont étudiés via des scénarios comprenant le comportement d'un utilisateur lambda lors de l'utilisation du programme. De cette manière, il est aisé de regrouper toutes les informations nécessaires à une fonctionnalité.

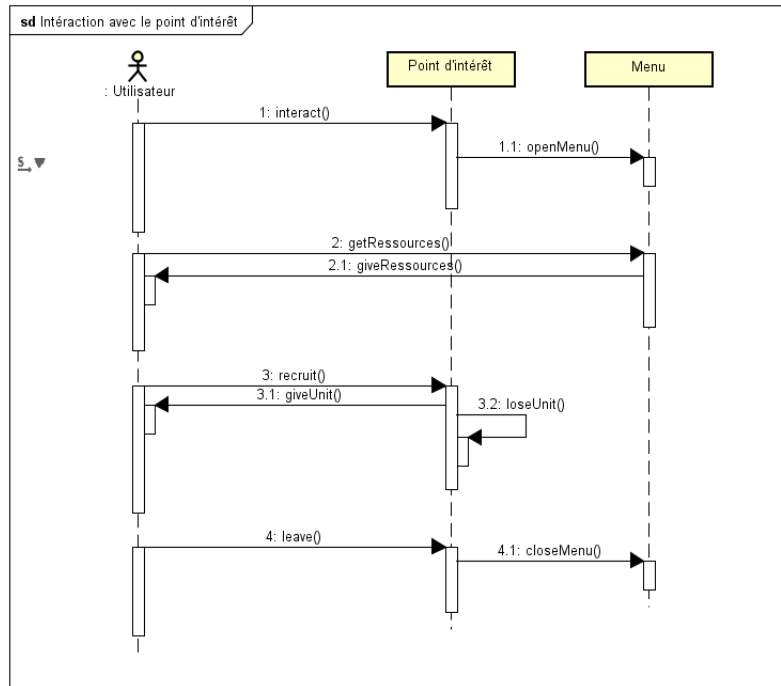
3.1.1 Diagrammes de séquence

Dans le cadre de ce projet, quelques diagrammes de séquences ont été créés pour les fonctionnalités de base.

Interaction avec un point d'intérêt

Ce premier montre le comportement simple d'un utilisateur souhaitant atteindre et inspecter un point d'intérêt. Sur ce point intérêt, il récupère des ressources, recrute une nouvelle unité et quitte le menu du point d'intérêt.

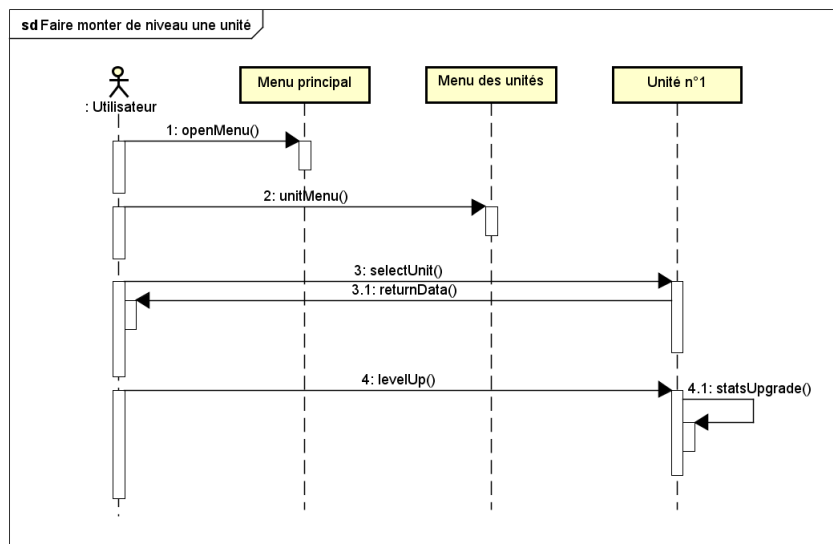
FIGURE 3.1 – Diagramme de séquence - Interaction avec un point d'intérêt



Montée de niveau d'une unité

Ce diagramme décrit un utilisateur améliorant une de ses unités. L'utilisateur ouvre le menu regroupant toutes ses unités. Il en sélectionne une et arrive sur son menu pour finalement la faire monter de niveau. Ses statistiques augmentent alors de quelques points.

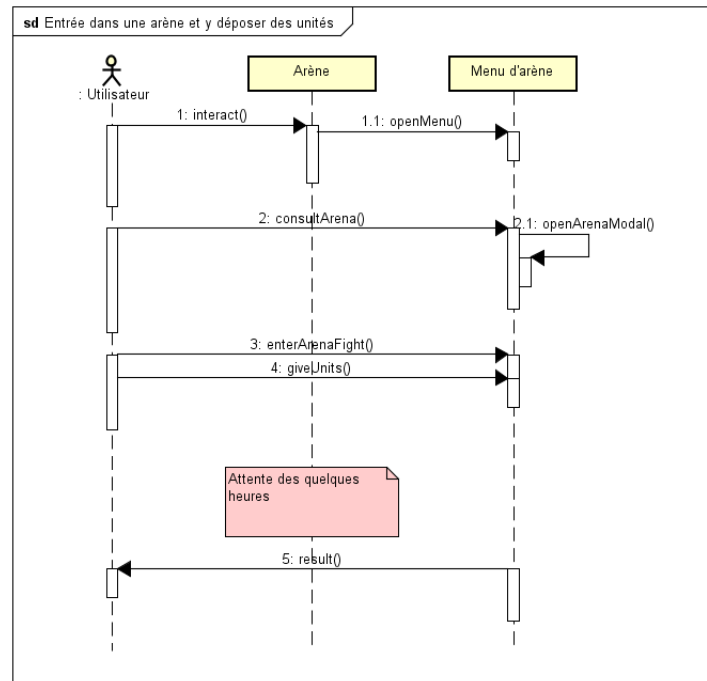
FIGURE 3.2 – Diagramme de séquence - Montée de niveau d'une unité



Entrée dans une arène

Ici, c'est l'entrée dans une arène qui est mise en valeur. L'utilisateur interagit avec l'arène ce qui ouvre un menu. Il décide alors de prendre d'assaut l'arène en y déposant des unités. On y montre une attente de quelques heures, c'est l'attente temps réel expliquée plus haut. Après cette attente, l'utilisateur reçoit une notification indiquant le résultat de la bataille et les ressources qu'il aura reçues en cas de victoire.

FIGURE 3.3 – Diagramme de séquence - Entrée dans une arène



3.2 Base de données

C'est lors de la réflexion de la conception qu'un problème est apparu. Le système de jeu que possède Camelote demande une persistance des données de l'utilisateur. Son château, ses ressources ainsi que ses actions doivent être retrouvés à l'identique ou dans un état avancé lorsqu'il rouvre l'application le jour d'après. Il existe plusieurs solutions à ce problème, voici celles qui ont été étudiées, puis celle qui a été retenue :

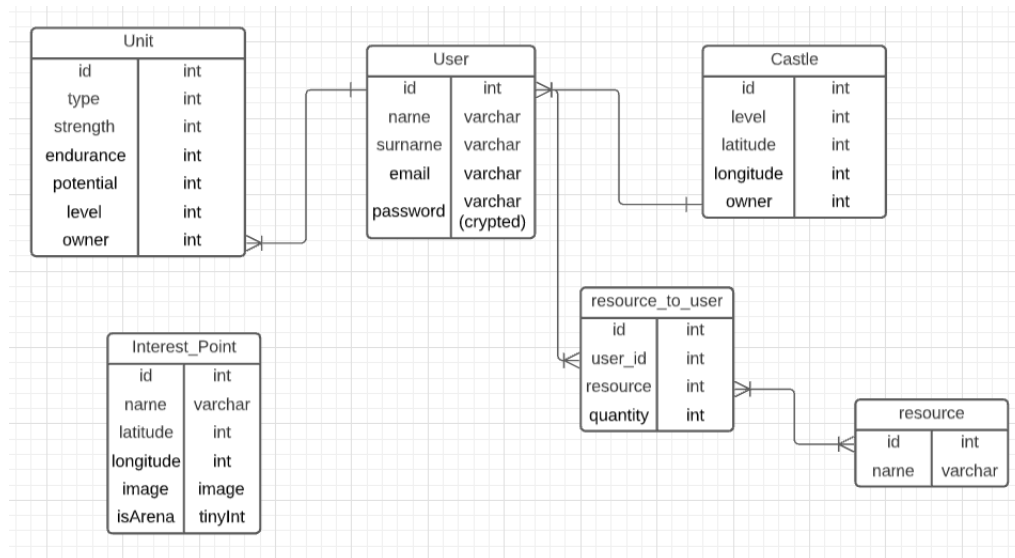
- La première solution étudiée a été le stockage de valeur en local, le problème est que l'espace de valeur locale lié à Unity n'est pas très grand et pas très pratique.
- La seconde est de créer un fichier dans l'espace disque du téléphone android pour y stocker les valeurs dont on a besoin. Le problème est que chaque téléphone possède un système de fichier différent et l'accès à un fichier précis n'est pas pratique.
- La dernière et celle qui a été choisie est de stocker les informations nécessaires dans une base de données.

Cette base de données est située sur un serveur, possédant un service phpmyadmin utilisable à distance, mis à disposition par l'école HE-Arc.

3.2.1 Schéma de base de données

Voici le schéma représentant les tables de la base de données.

FIGURE 3.4 – Schéma de base de données



Chaque utilisateur sera dans une table *User* où les informations générales comme le nom, le prénom ou encore l'email y sont placés. La table du château - *Castle* - est liée à l'utilisateur via un champ entier *owner* qui contiendra l'id du joueur correspondant. Les ressources sont liées au joueur via une table de "transition" qui permet de créer une relation N <-> N entre un utilisateur et les ressources. De *Resource* à *User* en passant par la table *resources_to_user*. Finalement, les unités de la table *Unit* sont liées au joueur de même manière que le château, via un champ *owner*.

Les points d'intérêts sont dans une table liée à rien possédant uniquement les informations de base et de position d'un point d'intérêt.

3.3 Structure de l'application

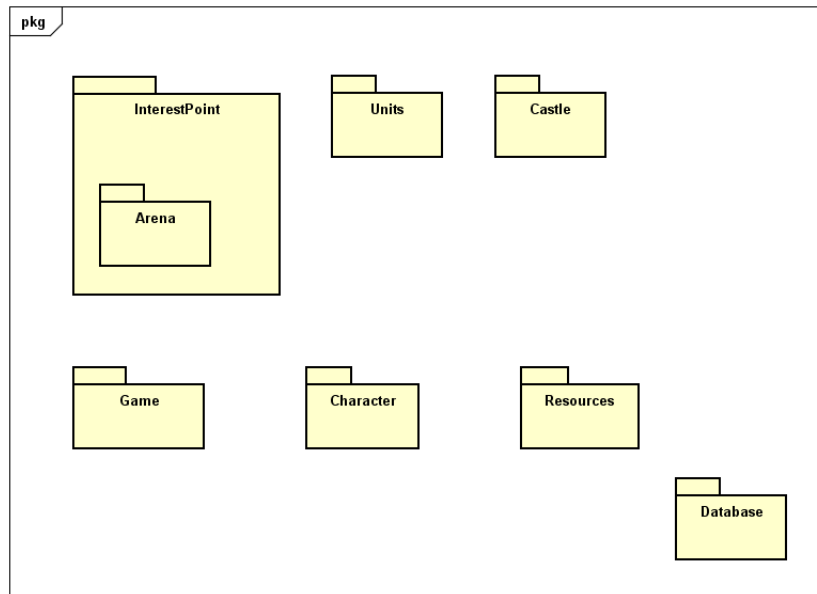
Le jeu Camelote est créé sur Unity et implémenté en C#. Dans le cas d'une création d'application standard, un diagramme de classe serait de mise : il permettrait de mettre en place la structure du code et de bien comprendre comment les classes et leurs membres sont définis. Cependant, Unity ne fonctionne pas avec des classes à proprement parler mais plus des "scripts" - Attention, on utilise ici un abus de langage, en effet, les scripts sont déclarés comme étant des classes mais sont utilisés comme de simples scripts, c'est-à-dire qu'un fichier représentera une fonctionnalité et peut des fois contenir une unique fonction.

3.3.1 Diagramme de package

C'est pourquoi, au lieu de créer un diagramme de classe, c'est un diagramme de package qui sera fourni. Ce diagramme comprendra une structure générale de l'application. Différents dossiers possédant des noms explicites sur leurs fonctions et ce qu'ils possèdent.

Les packages *InterestPoint*, *Units*, *Castle*, *Character* et *Resources* correspondent à des packages contenant des éléments ou des scripts correspondant aux éléments du jeu énoncés plus haut. Quant aux packages *Game* et *Database*, ils contiennent des scripts permettant de gérer le jeu. Le package *game* contiendra le *GameManager* qui lui gèrera les états du jeu et le package *Database* les scripts de communication avec la base de données.

FIGURE 3.5 – Diagramme de package



3.4 Langages, outils et frameworks utilisés

L'application a été créée via le moteur de rendu Unity et par conséquent implémenté en C#. De plus, comme expliqué ci-dessus le langage PHP est utilisé pour les scripts de communication avec la base de données. Finalement, le package UI Toolkit de Unity a été utilisé pour la création d'interfaces graphiques.

- Unity - Build Version 2021.2.7f1
- C#
- PhP - Version PHP 7.2.24-0ubuntu0.18.04.10
- PhPMyAdmin - Version 5.7.36-0ubuntu0.18.04.1

3.4.1 Bibliothèques et packages

- Unity - UI Toolkit - Version 1.0.0

Chapitre 4

Réalisation

La réalisation est la phase centrale d'un projet, c'est aussi celle qui va occuper le plus de temps. Ici, le sujet est l'implémentation des éléments importants du programme ainsi que leurs explications détaillées.

4.1 Managers

Lorsque l'on implémente un jeu, il faut créer ce qu'on appelle un "manager". Un manager est un élément qui va gérer le fil rouge du jeu, il va contenir les valeurs importantes ainsi que les états de l'application. Il suit le pattern Singleton pour être une unique instance et être accessible de partout. Camelote possède deux managers. Le premier est le "GameManager", il possède les états du jeu et gère les scènes de l'application. Le second est le "DBManager", c'est lui qui va gérer la communication avec la base de données et les appels de scripts correspondants.

4.1.1 GameManager

Le GameManager possède une enum correspondant aux états du jeu :

- Menu -> L'état du menu de lancement, avant de se connecter
- Logged -> L'état qui indique que l'utilisateur vient de se connecter
- Game -> L'état représentant la partie "en jeu"
- Castle -> L'état de la scène du château
- BackFromCastle -> L'état de retour de la scène du château

Il possède également l'id et le pseudo de l'utilisateur, ses ressources, sous la forme d'un dictionnaire qui a pour clé l'id de la ressource dans la base de données et pour valeur l'objet ressource, et finalement le château de l'utilisateur et la liste de ses unités.

Singleton

Le GameManager est un singleton qui est persistant entre les scènes. Comme c'est par cet objet que les scènes changent et que certaines fonctionnalités sont appelées, il est important d'y avoir accès partout.

```
1 // Singleton Instance
2 public static GameManager Instance;
3
4 // Awake is called before the start function
5 void Awake()
6 {
7     // If the instance exist then use it
8     if (Instance == null)
9     {
10         Instance = this;
11     }
12     else
13     {
14         Destroy(gameObject);
```

```

15         return;
16     }
17
18     // Line used to not destroy the GameManager when the user changes scene
19     DontDestroyOnLoad(gameObject);
20 }

```

Listing 4.1 – "Singleton et persistance entre les scènes"

La ligne "DontDestroyOnLoad(gameObject)" permet de faire en sorte que l'objet se retransmet aux prochaines scènes sans se détruire.

Changement d'état

Le changement d'état se fait via une fonction "public" prenant comme paramètre le nouvel état. Selon l'enum choisie, le GameManager lancera les actions correspondantes au nouvel état. C'est donc via cette fonction que les scènes du jeu sont lancées.

```

1  /// <summary>
2  /// Update the game state
3  /// </summary>
4  /// <param name="newState"></param>
5  public void UpdateGameState(GameState newState)
6  {
7      State = newState;
8
9      switch (newState)
10     {
11         case GameState.Menu:
12             break;
13         case GameState.Logged:
14             // Go to the main scene once the user is connected
15             SceneManager.LoadScene("MainGame");
16             // Load the user resources
17             DBManager.Instance.UserResources();
18             // Load the user castle
19             DBManager.Instance.UserCastle();
20             break;
21         case GameState.Game:
22             // The Google Maps service has started, the game can then begin
23             DBManager.Instance.LoadPOI();
24             break;
25         case GameState.Castle:
26             // The user decides to go the to Castle scene
27             SceneManager.LoadScene("Castle");
28             break;
29         case GameState.BackFromCastle:
30             // Change the scene from the castle to the game scene
31             SceneManager.LoadScene("MainGame");
32             break;
33         default:
34             throw new ArgumentOutOfRangeException(nameof(newState), newState, null);
35     }
36 }

```

Listing 4.2 – "Changement d'état et switch d'actions"

4.1.2 DBManager

Le DBManager est un Singleton qui est utilisé pour la communication avec la base de données. Sa seule fonctionnalité est de posséder les autres scripts C# qui sont liés à la base de données et les appeler via des fonctions public appelables depuis l'extérieur.

```

1  // Singleton instance
2  public static DBManager Instance;

```



```

3
4 // Scripts used to access the database
5 public LoadPointOfInterest lpof;
6 public UserRegister usrr;
7 public UserLogin usrl;
8 public UserResources usrr;
9 public UserCastle usrc;
10 public UpdateResources upr;
11 public UpdateCastle upc;
12 public InsertResources irs;
13
14 ...
15
16 /// <summary>
17 /// Create a new user
18 /// </summary>
19 public void UserRegister(string pseudo, string surname, string name, string email,
    string password, string checkPassword)
20 {
21     usrr.CallRegister(pseudo, surname, name, email, password, checkPassword);
22 }
23
24 /// <summary>
25 /// Log a user in
26 /// </summary>
27 public void UserLogin(string email, string password)
28 {
29     usrl.CallLogin(email, password);
30 }
31
32 ...

```

Listing 4.3 – "Manager du base de données"

4.2 Base de données

Pour accéder, modifier et insérer des valeurs dans la base de données [1], il faut pouvoir communiquer avec celle-ci. En reprenant l'exemple du projet de l'ancien élève, il a été décidé de reprendre sa manière de faire et de communiquer via des scripts PHP utilisés comme des Web Services. En effet, plusieurs scripts, un script pour une fonctionnalité, sont stockés sur un serveur héberger par l'école.

4.2.1 Scripts PHP

Tous les scripts php ont utilisés *mysqli* [2] [3] [4] pour la connexion à la base de données et si la requête retourne un objet, alors il est encodé sous format json pour facilement le récupérer du côté de Unity.

Récupération des points d'intérêts

La récupération des points d'intérêts est un script très simple où un simple "SELECT" est employé et chaque résultat est traité et ajouté dans un tableau contenant tous les points d'intérêts du jeu. Finalement, le tableau entier est encodé en json et est affiché via un echo pour être récupéré par Unity par la suite.

```

1 // Initialise the json array
2 $poi = array();
3 // Query which will get all the interest_points
4 $poi_query = "SELECT * FROM interest_point";
5 // Get the data from the query
6 if ($result = $database->query($poi_query)) {
7     // If data is found, store it inside an array
8     while($row = $result->fetch_assoc()) {
9         $poi[] = $row;

```

```

10 }
11 // Return the array encoded in json
12 echo json_encode($poi);
13 } else {
14 // No point of interests, send 1 as an error
15 echo "-1";
16 }

```

Listing 4.4 – Récupération des ressources

Connexion et inscription

La connexion et l'inscription d'utilisateur sont très simples :

- La connexion récupère en input via le verbe "POST" l'email et le mot de passe de l'utilisateur pour vérifier s'il existe un compte possédant cette adresse. Si oui, alors il compare les deux mots de passe, crypté bien entendu, et s'ils correspondent, alors il accepte la connexion.
- L'inscription récupère en input via le verbe "POST" toutes les informations nécessaires à l'utilisateur et vérifie si l'adresse email n'est pas déjà utilisée. Sinon, elle va insérer dans la base de données un utilisateur nouveau avec les informations retenues. De plus, un château sera créé pour lui.

La particularité de ces requêtes est qu'elles n'ont pas besoin d'être encodées en json. En effet, le résultat attendu n'est pas un objet ! Pour la connexion, uniquement l'id et le pseudo sont nécessaires et quant à l'inscription, aucun retour n'est attendu.

Ressources

Il existe trois scripts php pour les ressources. Un permettant de les récupérer via un "SELECT", le second pour les mettre à jour et le dernier pour en insérer de nouvelles. La récupération de ressources fonctionne de la même manière que les points d'intérêt. On récupère tous les résultats et on les stocke dans un tableau que l'on encode en json avant de l'afficher comme pour le script précédent.

Quant aux deux autres, ils fonctionnent de manière un peu spécifique.

L'insertion va récupérer en premier lieu l'id de l'utilisateur reçu via "POST". C'est avec cette valeur que la ressource sera attribuée à un propriétaire. Ensuite, les clés du tableau "\$_POST" sont récupérées, elles correspondent à l'id de la ressource qui va être ajoutée. Quant à la valeur liée à la clé, elle sera la quantité de la nouvelle ressource. Finalement, la boucle commence à l'indice 1 pour éviter de récupérer la clé 'id'.

```

1 // User's id
2 $id = $_POST['id'];
3
4 // Get all the keys, they represent which resource we need to insert, the value is
   its new amount
5 $keys = array_keys($_POST);
6
7 // Loop on all the resources which need to be inserted
8 for($i = 1; $i < count($keys); $i++) {
9     $resources_query = "INSERT INTO resources_to_user (user_id, resources,
   quantity) VALUES (?, ?, ?)";
10    $stmt = $database->prepare($resources_query);
11    $stmt->bind_param("iii", $id, $keys[$i], $_POST[$keys[$i]]);
12    $stmt->execute();
13 }

```

Listing 4.5 – Insertion d'une nouvelle ressource

La mise à jour ressemble beaucoup à l'insertion mise à part un point. Lors de la requête "UPDATE", ce n'est pas l'id de la ressource qui est utilisée, mais l'id de l'entrée dans la table *resources_to_user*. De cette manière, il n'y a pas besoin de savoir quelle ressource est modifiée, juste sa future valeur et à quel endroit de la table elle est située.

```

1 // Get all the keys, they represent which resource we need to update, the value is
  its new amount
2 $keys = array_keys($_POST);
3
4 // Loop on all the resources which need to be updated
5 for($i = 0; $i < count($keys); $i++) {
6     $update_query = "UPDATE resources_to_user SET quantity=? WHERE id=?";
7     $stmt= $database->prepare($update_query);
8     $stmt->bind_param("ii", $_POST[$keys[$i]], $keys[$i]);
9     $stmt->execute();
10 }

```

Listing 4.6 – Mise à jour d’une ressource

Château

Pour ce qui est du château, il n’existe que deux scripts, un qui récupère l’état actuel du château de l’utilisateur et l’autre qui le met à jour lorsqu’il monte de niveau.

4.2.2 Liaison entre unity et les scripts php

Bien que les scripts PHP soient créés et fonctionnels, il faut pouvoir les atteindre depuis le code du jeu, depuis Unity. Pour ce faire, des scripts C#, qui se connectent à l’adresse IP du serveur, lancent les scripts correspondants et récupèrent les informations nécessaires en retour. Voici un exemple simple d’un script C# qui va récupérer l’état du château de l’utilisateur.

La première chose qui se fait est la création d’un objet WWWForm qui va accueillir les paramètres "POST" envoyé au script php. La connexion au serveur fonctionne via un objet "UnityWebRequest" provenant du *namespace* "UnityEngine.Networking". Le procédé se fait de manière asynchrone pour ne pas bloquer l’application lorsque l’information est entrain d’être récupéré/traité par l’objet. Une fois la requête traitée, on récupère le résultat, si une erreur s’est produite on l’affiche, sinon on traite le résultat. Ici, on récupère le château sous format json et on le transforme en objet pour finalement le stocker dans le GameManager. De cette manière, l’état du château est conservé pour la session de jeu.

```

1 /// <summary>
2 /// Send a request to get the user's castle
3 /// </summary>
4 /// <returns></returns>
5 IEnumerator GetCastle()
6 {
7     // Form used to send the user id to the php script
8     WWWForm form = new WWWForm();
9     form.AddField("id", GameManager.Instance.Id);
10
11     // Create and send the post request to the php script
12     UnityWebRequest www = UnityWebRequest.Post("http://157.26.64.176/camelote/
    UserCastle.php", form);
13     yield return www.SendWebRequest();
14
15     // If there is an error, return it
16     if (www.result == UnityWebRequest.Result.ConnectionError || www.result ==
    UnityWebRequest.Result.ProtocolError)
17     {
18         Debug.Log(www.error);
19     }
20     else
21     {
22         // Get the data in JSON form
23         string data = www.downloadHandler.text;
24         if (data != "-1")
25         {
26             string jsonString = JsonHelper.FixJson(data);
27             this.userCastle = JsonHelper.FromJson<Castle>(jsonString)[0];
28             // Get the user's castle

```

```

29         GameManager.Instance.Castle = this.userCastle;
30     }
31     else
32     {
33         Debug.Log("No resources found");
34     }
35 }
36 }

```

Listing 4.7 – "Récupération de l'état actuel du château"

Finalement, cette fonction est appelée dans une coroutine pour permettre à Unity de gérer le tout de manière asynchrone.

```

1 public void CallCastle()
2 {
3     StartCoroutine(GetCastle());
4 }

```

Listing 4.8 – "Appel asynchrone de la fonction GetCastle()"

Les autres scripts de communication fonctionnent de la même manière, les seules différences sont les traitements des paramètres avant l'appel du script php et le traitement des résultats.

4.3 Interface graphique

Sur Unity, il existe trois manières principales de créer des interfaces graphiques.

- Unity UI
 - Consiste à utiliser des éléments "canvas" de Unity et de le remplir d'éléments par exemple du texte ou des images. Est simple mais un peu rigide
- IMGUI
 - IMGUI n'est pas fait pour créer des interfaces graphiques au Runtime, c'est-à-dire pour le jeu. Il est fait pour créer des interfaces graphiques pour l'éditeur, si l'on souhaite créer nos propres menus par exemple.
- Unity UI Toolkit
 - Unity UI Toolkit est un système de création d'interfaces graphiques utilisant le format UXML. Un format XML modifié pour Unity. Très pratique pour créer de belles interfaces graphiques mais difficile à prendre en main

Dans l'objectif d'apprendre un nouveau penchant de Unity, il a été décidé d'explorer et d'essayer le système Unity UI Toolkit pour créer des interfaces graphiques. Le package offre un système de création à la manière d'un UI Builder de Qt. Via des *Drag and drop*, on peut placer les éléments souhaités et en modifier leurs valeurs dans un inspecteur.

L'unique problème de UI Toolkit est qu'il est difficile à prendre en main. En effet, la documentation de Unity UI Toolkit est très légère et il est compliqué de trouver d'exemple dit "propre" sur internet. Les interfaces décrites ci-dessous ont été réalisées avec le peu d'informations trouvées. [5] [6] [7] [8] [9]

4.3.1 UI Toolkit

Les interfaces graphiques du jeu sont séparées en deux parties distinctes. Les interfaces du menu de lancement et les interfaces du jeu. Celles de la première catégorie sont celles qui servent à l'utilisateur à se connecter et à s'inscrire et celles de la seconde, à accéder aux fonctionnalités du jeu comme aller voir ses ressources ou encore accéder à son château.

Notion de UIManager et de scripts d'interface

UI Toolkit fonctionne via un système de UI Document, un GameObject placé dans la scène du jeu qui contient l'élément UXML à afficher. Seulement, un UI Document ne possède qu'un unique emplacement pour l'affichage. Pour faire fonctionner un menu, un affichage qui se déplace entre

différentes interfaces selon les besoins, il faut pouvoir appliquer un "changement" d'affichage. Ce faisant, UI Toolkit offre plusieurs manières de faire, dont deux ont été étudiées :

- Instancier toutes ces interfaces graphiques dans la scène, créer un script pour chaque interface et lier une fonction aux boutons affichant ou cachant l'interface souhaitée.
- Créer un script qui déclare une classe interne UXML factory et qui hérite de Visual Element pour se transformer en base d'interface graphique. De cette manière on peut créer une interface graphique en son sein et le script possédera ses éléments pour interagir directement avec eux. On crée alors ce qui s'appelle un UI Manager, un script qui contient toutes les autres interfaces graphiques et les affiche selon les actions correspondantes.

Après avoir testé la première possibilité jugée rigide et peu flexible, c'est via les UI Manager que les interfaces de Camelote ont été affichées. Ci-dessous se trouve le code de l'UI Manager.

```
1 public class MainMenuManager : VisualElement
2 {
3     // All the different interfaces which will be switched to
4     VisualElement startInterface;
5     VisualElement loginInterface;
6     VisualElement registerInterface;
7
8     // Public class used to get the MainMenuManager UXML
9     public new class UxmlFactory : UxmlFactory<MainMenuManager, UxmlTraits> { }
10
11     public MainMenuManager()
12     {
13         // Register to the GeometryChangedEvent
14         this.RegisterCallback<GeometryChangedEvent>(OnGeometryChange);
15     }
16 ...
```

Listing 4.9 – "Définition d'un UI Manager"

La classe hérite donc de VisualElement et non de MonoBehaviour, elle devient une interface graphique à part entière. On peut y voir la déclaration de la classe UxmlFactory, c'est via cette classe que le script se transforme en une interface graphique utilisable via Unity UI Toolkit. De plus, elle possède les membres "interfaces" de type VisualElement représentant une interface graphique. En effet, comme dit plus haut, un UI Manager contient toutes les autres interfaces du menu ! C'est via ces variables que le manager pourra afficher ou modifier les affichages des différents menus.

Cependant, il y a un détail important. Dans le constructeur, le manager s'enregistre au callback de la fonction "OnGeometryChange". Cette fonction peut être comparée au "OnEnable" des scripts MonoBehaviour, ce n'est pas une fonction appelée à la construction, mais à "l'activation". Pour les objets VisualElement, "OnGeometryChange" est appelé à chaque changement de taille, transformation géométrique et surtout à la première activation de l'interface, juste après que ses composants aient été instanciés ! C'est dans cette fonction que les éléments graphiques seront récupérés et les fonctions attribuées.

```
1 void OnGeometryChange(GeometryChangedEvent evt)
2 {
3     // Set all the interfaces to the children objects
4     startInterface = this.Q("StartInterface");
5     loginInterface = this.Q("LoginInterface");
6     registerInterface = this.Q("RegisterInterface");
7
8     // Set the functions to the main menu interface buttons
9     startInterface?.Q("login-button")?.RegisterCallback<ClickEvent>(ev =>
LoginScreen());
10     startInterface?.Q("register-button")?.RegisterCallback<ClickEvent>(ev =>
RegisterScreen());
11
12     // Set the function to the back to the main menu button
13     loginInterface?.Q("back-button")?.RegisterCallback<ClickEvent>(ev =>
BackToMainMenu());
14     registerInterface?.Q("back-button")?.RegisterCallback<ClickEvent>(ev =>
BackToMainMenu());
15
```

```

16 // Unregistrer from the event, so it can trigger only once (at the beginning)
17 this.UnregisterCallback<GeometryChangedEvent>(OnGeometryChange);
18 }

```

Listing 4.10 – "Exemple de OnGeometryChange"

Maintenant que la notion de UI Manager est expliquée, voici les implémentations et les spécificités des différentes interfaces graphiques de Camelote.

Menu de lancement

Le menu de lancement est géré par le "MainMenuManager", l'UI Manager qui s'occupe du menu de lancement du jeu. C'est via ce script que l'interface change entre le menu de connexion, le menu d'inscription de compte et le lancement du jeu.

Ces menus sont très simples. Le menu de connexion est un menu formulaire qui possède deux champs, un pour l'adresse email et l'autre pour le mot de passe. Celui d'inscription est similaire, à l'exception des champs qui eux sont présents pour récupérer les informations de l'utilisateur comme son nom et son prénom.

Une fois l'utilisateur connecté, le jeu se lance.

Menu de jeu

Le menu de jeu ressemble beaucoup au menu de lancement, il est géré par l'"UIManager" qui va s'occuper d'ouvrir le menu via le bouton en forme de bouclier, d'aller sur la liste des ressources ou sur le château de l'utilisateur. Il va également s'occuper d'afficher l'interface d'interaction avec les points d'intérêts disposés sur la carte.

Liste des ressources

La liste des ressources est une interface un peu particulière, elle possède un élément nommé "ListView" qui va gérer une liste d'éléments de manière dynamique. Cet objet permet de contenir les éléments affichant les ressources et la quantité qu'en possède le joueur. Chaque *item* présent dans la liste est une instance d'un objet source, appelé "Ressource", qui a pour seul contenu deux labels : un pour le nom de la ressource et l'autre pour la quantité.

Pour comprendre son fonctionnement [10], il faut montrer quelques bouts de code.

```

1 public class UIResources : VisualElement
2 {
3     // ListView element
4     private ListView resourcesList;
5     private List<Resource> resources;
6
7     // Public class used to get the UIManager UXML
8     public new class UxmlFactory : UxmlFactory<UIResources, UxmlTraits> { }
9
10    public UIResources()
11    {
12        UserResources.OnRefresh += RefreshData;
13        this.RegisterCallback<GeometryChangedEvent>(OnGeometryChange);
14    }
15    ...

```

Listing 4.11 – "Définition du script de la ListView"

Le script possède un objet ListView qui sera la liste à remplir ainsi qu'une liste de ressource qui correspond aux ressources stockées dans le "Dictionnaire" de l'utilisateur. Ici, une liste est utilisée, car comme son nom l'indique, ListView fonctionne via l'interface IList. La récupération de ces ressources dans une liste est expliquée plus bas. On y voit également l'inscription à un événement "OnRefresh" de la classe UserResources. Cet événement est *trigger* à chaque fois que l'utilisateur va mettre à jour sa liste de ressources, de cette manière, la méthode "RefreshData" sera également appelée et mettra à jour l'élément ListView.

L'objet ListView fonctionne de cette manière :

- Il a besoin d'une fonction "MakeItem" qui va créer l'*item* de type VisualElement présent dans la liste.
- D'une fonction "BindItem" qui va lier la/les donnée(s) souhaitée(s) à l'élément créé ci-dessus.
- Il peut accepter une valeur "fixedItemHeight" qui attribue une valeur taille aux *items* de la liste, cependant elle est subsidiaire et ne sert qu'à gérer le style.
- Et finalement d'une liste "itemsSource" qui est le conteneur des données à utiliser, il doit impérativement implémenter l'interface IList pour être accepté.

```

1 void OnGeometryChange(GeometryChangedEventArgs evt)
2 {
3     this.resourcesList = this.Q<ListView>("resources-list");
4
5     // Create the list view for the resources
6     if (GameManager.Instance != null && GameManager.Instance.Resources != null)
7     {
8         this.resources = DictToList(GameManager.Instance.Resources);
9         this.resourcesList.makeItem = MakeItem;
10        this.resourcesList.bindItem = BindItem;
11        this.resourcesList.itemsSource = this.resources;
12        this.resourcesList.fixedItemHeight = 30;
13    }
14
15    // Unregistrer from the event, so it can trigger only once (at the beginning)
16    this.UnregisterCallback<GeometryChangedEventArgs>(OnGeometryChange);
17 }

```

Listing 4.12 – "Mise en place des valeurs de la ListView"

Ci-dessous sont décrites les fonctions "MakeItem" et "BindItem" utilisées et expliquées plus haut.

```

1 private VisualElement MakeItem()
2 {
3     //Here we take the uxml and make a VisualElement
4     VisualTreeAsset resourceItem = Resources.Load<VisualTreeAsset>("
5     ResourceElement");
6     VisualElement listItem = resourceItem.Instantiate();
7     return listItem;
8 }
9 private void BindItem(VisualElement e, int index)
10 {
11     //We add the resource name to the label of the list item
12     e.Q<Label>("name-label").text = this.resources[index].name;
13     //We add the resource quantity to the label of the list item
14     e.Q<Label>("quantity-label").text = this.resources[index].quantity.ToString();
15 }

```

Listing 4.13 – "MakeItem et BindItem"

Finalement, la fonction de traduction du dictionnaire à la liste et la fonction servant à rafraîchir le contenu de la liste. La première récupère toutes les valeurs du dictionnaire n'étant pas *null* et les ajoute à la liste et la seconde récupère l'état actuel des ressources de l'utilisateur et met à jour la ListView en appelant la fonction "Rebuild".

```

1 private List<Resource> DictToList(Dictionary<int, Resource> resources)
2 {
3     List<Resource> list = new List<Resource>();
4     foreach (KeyValuePair<int, Resource> kvp in resources)
5     {
6         if (kvp.Value != null)
7         {
8             list.Add(kvp.Value);
9         }
10    }
11    return list;
12 }
13
14 public void RefreshData()

```

```

15 {
16     if (this.resourcesList != null)
17     {
18         this.resources = DictToList(GameManager.Instance.Resources);
19         this.resourcesList.itemsSource = this.resources;
20         this.resourcesList.Rebuild();
21     }
22 }

```

Listing 4.14 – "Fonctions utilitaires pour la ListView"

Château

On accède à la scène du château au travers du bouton "Château" du menu, ce bouton va simplement appeler un changement d'état du GameManager et ainsi appeler un changement de scène pour arriver face au château de l'utilisateur. L'interface de cette scène est très simple, elle possède deux boutons, un qui permet de revenir en arrière et l'autre qui fait monter de niveau le château, à condition d'avoir suffisamment de ressources. Finalement, un regroupement de labels permettant de connaître combien de ressources l'utilisateur possède par rapport à la quantité nécessaire à l'amélioration est situé en haut à droite de l'écran.

Interaction

Lorsque l'on clique sur un point d'intérêt, l'"UIManager" fait apparaître l'interface d'interaction. Elle comprend les informations du point comme son nom, sa position et un bouton qui permet de récolter les ressources disponibles.

4.4 Points d'intérêts et ressources

Les points d'intérêts et les ressources sont tous les deux des éléments qui sont récupérés et chargés depuis la base de données au tout début de la session de jeu. En effet, lors du changement d'état indiquant la connexion et le lancement du jeu, le GameManager appelle le DBManager pour récupérer les valeurs sur la base de données et les stocker. Comme dit plus haut, les ressources sont stockées et gérées dans un dictionnaire situé dans le GameManager. Quant aux points d'intérêts, ils sont placés et créés comme étant des enfants de la "MapsSDK" l'élément qui correspond à la carte Google Maps. Pour leurs créations, le script "LoadPointOfInterest" utilise un *prefab* d'un objet 3D, ressemblant à un point d'exclamation, qui est l'apparence des points d'intérêts dans le jeu. Le script y stocke alors les informations du point et le place sur la carte via la position reçue.

Le prefab du point d'intérêt possède un *component* "InteractionPOI" qui gère le clique ainsi que la génération des ressources. Si l'utilisateur interagit avec le point d'intérêt en le touchant, le script "InteractionPOI" va *trigger* un *event* pour que l'"UIManager" affiche l'interface décrite plus haut. Vient donc la partie la plus importante du point d'intérêt : la génération des ressources.

Ce code se trouve dans le script de l'interface graphique de l'interaction, il va générer deux ressources aléatoires ainsi qu'une quantité aléatoire pour finalement mettre à jour la base de données.

```

1 void GetResources()
2 {
3     // Lists which will contain the new resources data
4     List<int> resourcesIDs = new List<int>();
5     List<int> resourcesAmounts = new List<int>();
6
7     string informationText = "";
8
9     // Resource max amount
10    int max = GameManager.Instance.Castle.level * 10;
11
12    for (int i = 0; i < 2; i++)
13    {
14        // Generate the random resource
15        int resourceId = Random.Range(1, 7);

```



```

16         // Generate the random amount
17         int resourceAmount = Random.Range(5, max + 1);
18         if (GameManager.Instance.Resources[resourceId] == null)
19         {
20             // Add it to the "to be inserted" list
21             resourcesIDs.Add(resourceId);
22             resourcesAmounts.Add(resourceAmount);
23         }
24         else
25         {
26             GameManager.Instance.Resources[resourceId].quantity += resourceAmount;
27         }
28
29         informationText += this.ResourcesNames[resourceId] + " : " +
resourceAmount + "\n";
30     }
31
32     // Show what the user got
33     this.Q<Label>("info-resources-label").text = informationText;
34     this.Q<Label>("info-resources-label").style.display = DisplayStyle.Flex;
35
36     // Update database
37     DBManager.Instance.UpdateResources();
38     if (resourcesIDs.Count > 0)
39     {
40         DBManager.Instance.InsertResources(resourcesIDs, resourcesAmounts);
41     }
42 }

```

Listing 4.15 – "Génération des ressources"

4.5 Château

Comme pour les ressources et les points d'intérêts, le château est récupéré de la même manière au début de la session de jeu. La scène possède trois prefabs différents de château, un pour chaque niveau possible : de 1 à 3. Le script "CastleManagement" s'occupe de toutes les tâches liées au château :

- Son amélioration
- La mise à jour de son prefab
- Il vérifie également si l'utilisateur possède suffisamment de ressources avant de l'améliorer

Pour l'amélioration la fonction "UpgradeCastle" vérifie si les montants sont corrects et appelle la fonction "SpendResources" du GameManager : c'est elle qui va dépenser les ressources directement. Si les quantités étaient correctes, le château monte de niveau, les données stockées sur la base de données sont mises à jour et le prefab est changé.

```

1 public bool UpgradeCastle() {
2     // If there is enough resources
3     if(EnoughResources()) {
4         // Spend them
5         GameManager.Instance.SpendResources();
6     } else {
7         return false;
8     }
9     // Upgrade the castle level
10    GameManager.Instance.Castle.level++;
11    // Call the database scripts to upgrade the castle level and update the
resources' amounts
12    // Update the castle's level in the database
13    DBManager.Instance.UpdateCastle();
14    // Update the resources in the database
15    DBManager.Instance.UpdateResources();
16    // Change the castle appearance
17    UpdateCastle();
18    return true;

```

Listing 4.16 – "Génération des ressources"

Cette méthode est appelée via l'interface décrite plus haut.

4.6 Tests

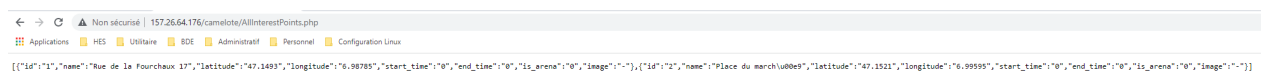
L'application a beaucoup été testée au fil de son implémentation. Comme beaucoup de fonctionnalités se basent sur des scripts "externes", c'est-à-dire les scripts php, il a d'abord fallu les tester avant de les employer directement. De plus, Unity UI Toolkit n'est pas très adapté pour des jeux mobiles, c'est pourquoi de nombreux *builds* ont été créés sur un téléphone prêté par l'école.

4.6.1 Tests des différents scripts php

Les scripts phps sont situés sur un serveur de l'école dont l'accès a été ouvert le temps du projet. Pour les tester, il fallait y accéder via l'adresse IP du serveur avec le chemin correspondant. Les messages retournés via "echo" indiquaient alors si le script fonctionnait comme prévu.

Dans le cas du script d'inscription, si l'utilisateur existe déjà dans la base de données, un "-1" est alors écrit sur la page du script et si l'insertion se fait correctement alors rien n'est affiché. Pour ce qui est des scripts qui vont chercher des valeurs comme le "SELECT" des points d'intérêts, alors c'est directement le résultat qui y est écrit.

FIGURE 4.1 – Récupération des points d'intérêts



4.6.2 Builds multiples

L'édition d'interface via Unity UI Toolkit fonctionne d'une manière assez étrange. La taille de l'interface lors de sa construction n'est en rien importante, elle s'adaptera à l'écran du jeu ; à la caméra. Même si on lui donne une taille fixe correspondant à la résolution de l'écran de téléphone, il se peut que les rapports ne soient pas gardés. C'est ce qui est arrivé lors des tests de l'application.

La totalité des interfaces graphiques, qui au premier abord étaient de la bonne taille sur l'éditeur Unity, étaient complètement déformées une fois sur le téléphone. Il a fallu alors tester des changements de taille à l'aveugle et faire des builds pour adapter les interfaces graphiques au téléphone physique.

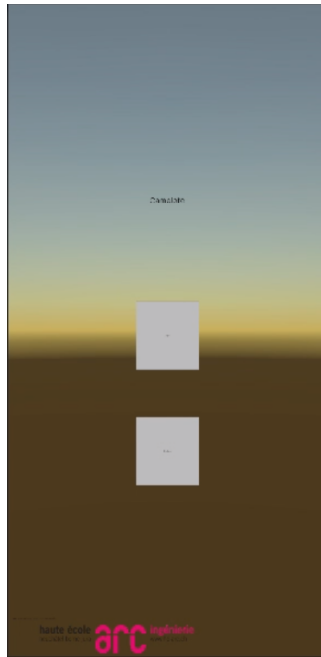


FIGURE 4.2 – Le visuel dans l'éditeur

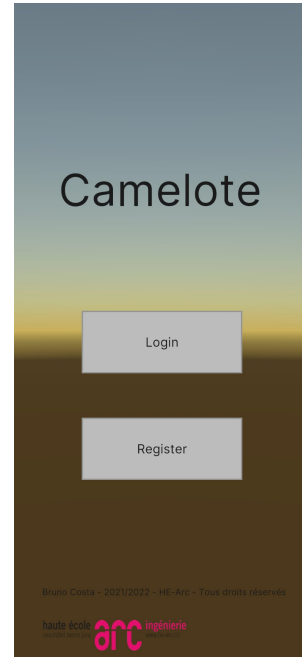


FIGURE 4.3 – Le visuel sur le téléphone

De plus, à l'aide de ces builds les différentes fonctionnalités de l'application ont été testées à maintes reprises.

Chapitre 5

Résultats

Camelote est un projet conséquent, en effet, de la création d'interfaces graphiques, une communication avec une base de données, utilisation de l'APK de Google Maps et des éléments de gameplay avec une interaction sur des points d'intérêts et des gestions d'unités, l'application représente un jeu complet et intéressant à créer. C'est pourquoi, au début du module, les objectifs ont été séparés en parties distinctes, primaires, secondaires et tertiaires.

Dans ce chapitre seront listés les objectifs atteints et les non atteints ainsi que l'état actuel de l'application.

5.1 Objectifs atteints

5.1.1 Objectifs primaires

1. Prise en main du projet initial.
 - Le projet initial a été analysé et "déconstruit" pour ne récupérer que ce qui est utile au nouveau projet.
2. Modification de certaines fonctionnalités du projet initial pour correspondre au jeu final.
 - La modification a été apportée principalement sur l'interface graphique, sinon, la plupart des fichiers correspondaient déjà à ce qui était souhaité.
3. Gestion de la base de données.
 - L'application communique avec la base de données et insère, met à jour et sélectionne les éléments et valeurs de l'application.
4. Implémentation du système de points d'intérêts ainsi que des différentes ressources.
 - Les points d'intérêts sont disponibles sur la carte et l'utilisateur peut interagir avec et récupérer des ressources via ceux-ci. De plus, il peut visionner ses ressources sur le menu correspondant du jeu.
5. Implémentation de l'interface graphique.
 - Les différentes interfaces sont présentes et implémentées.
6. Mécanique de jeu fonctionnelle : Gestion du château, PNJs.
 - La mécanique de gestion du château est présente et fonctionnelle.

5.2 Objectifs non-atteints

5.2.1 Objectifs secondaires

1. Mécanique de jeu fonctionnelle : Gestion du château, PNJs.
 - L'apparition de PNJs qui ont pour fonction d'être marchands n'a pas pu être implémentée.
2. Implémentation du système de bataille.
 - Les arènes et leurs systèmes de bataille n'ont pas été implémentés.
3. Recrutement des unités.
 - Les unités n'ont pas été implémentées et du coup le recrutement non plus.

4. Améliorations des unités.
 - L'amélioration des unités n'a pas été implémentée.

5.2.2 Objectifs tertiaires (Nice to have)

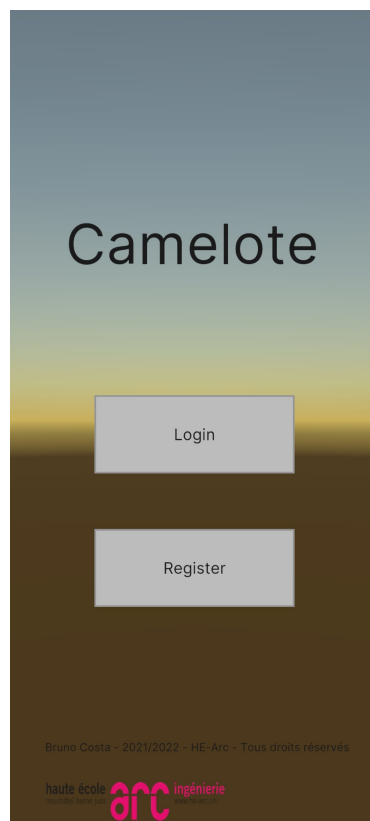
1. Système de textures médiévales appliquées procéduralement.
 - Les textures des bâtiments sont les textures du projet initial.
2. Bon design 3D des différents éléments.
 - Les designs de l'application sont ceux de bases.
3. Système multijoueur à la Pokemon GO.
 - Le système en ligne n'est pas implémenté.
 - Par conséquent, on ne peut pas inspecter d'autres joueurs ni se battre en arène en multijoueur.

5.3 État actuel de l'application

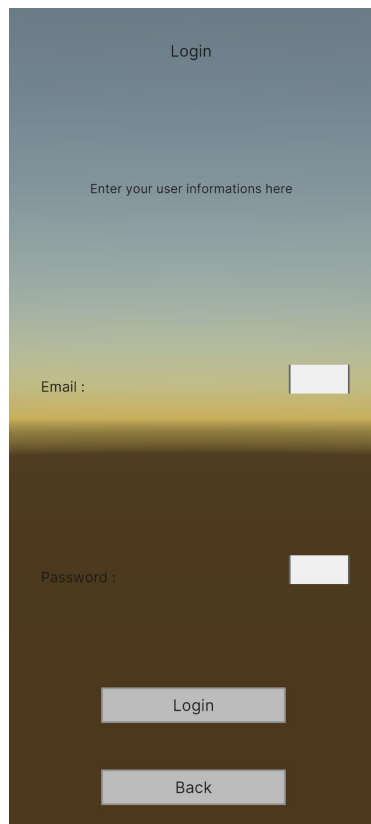
L'application est à présent dans un état stable et fonctionnel. La quasi-totalité des objectifs primaires a été respectée et les objectifs secondaires ne l'ont pas été par manque de temps. Voici le visuel de l'état actuel de l'application.

5.3.1 Menu principal

FIGURE 5.1 – Menu principal du jeu

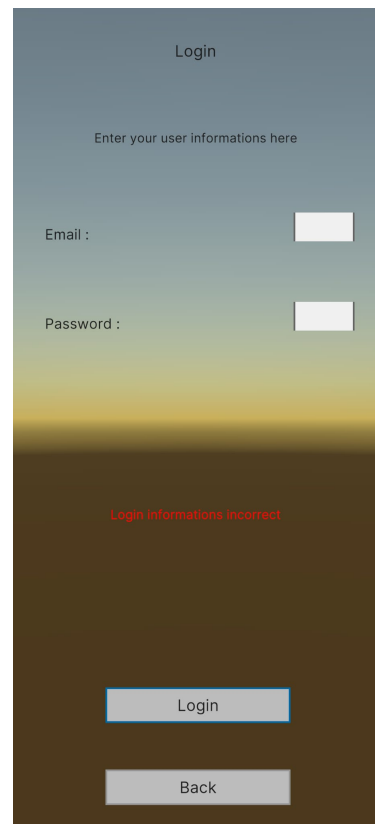


5.3.2 Connexion et inscription



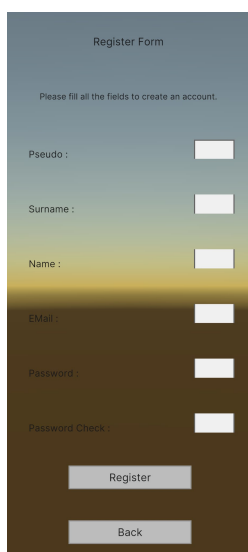
A mobile app login screen with a dark blue header and a brown gradient background. The title 'Login' is at the top. Below it is the instruction 'Enter your user informations here'. There are two input fields: 'Email :' and 'Password :'. At the bottom are two buttons: 'Login' and 'Back'.

FIGURE 5.2 – Menu de connexion



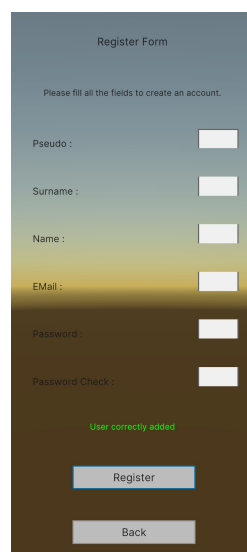
The same login screen as Figure 5.2, but with a red error message 'Login informations incorrect' displayed below the input fields. The 'Login' button is highlighted with a blue border.

FIGURE 5.3 – Erreur dans le menu de connexion



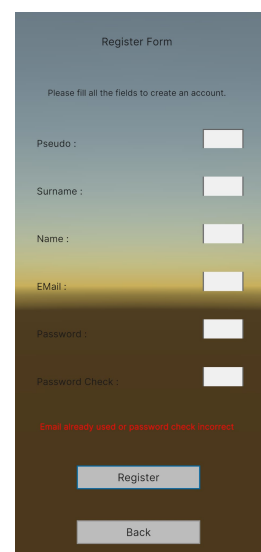
A mobile app registration screen with a dark blue header and a brown gradient background. The title 'Register Form' is at the top. Below it is the instruction 'Please fill all the fields to create an account.'. There are six input fields: 'Pseudo :', 'Surname :', 'Name :', 'EMail :', 'Password :', and 'Password Check :'. At the bottom are two buttons: 'Register' and 'Back'.

FIGURE 5.4 – Menu d'inscription



The same registration screen as Figure 5.4, but with a green success message 'User correctly added' displayed below the input fields. The 'Register' button is highlighted with a blue border.

FIGURE 5.5 – Inscription de l'utilisateur correcte



The same registration screen as Figure 5.4, but with a red error message 'Email already used or password check incorrect' displayed below the input fields. The 'Register' button is highlighted with a blue border.

FIGURE 5.6 – Erreur dans l'inscription de l'utilisateur

5.3.3 Interface du jeu

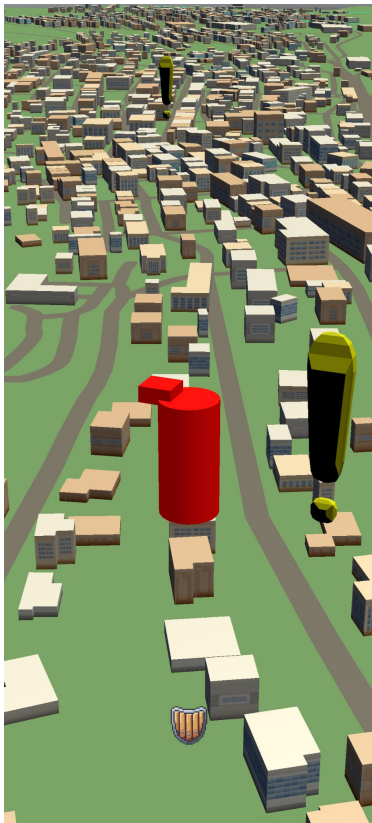


FIGURE 5.7 – Interface de jeu

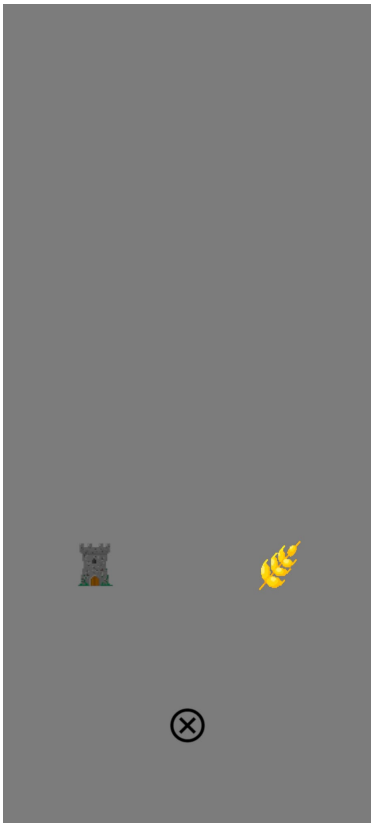


FIGURE 5.8 – Menu du jeu ouvert

FIGURE 5.9 – Menu des ressources

< Resources	
Blé	201
Fer	29
Pierre	17
Nourriture	272
Bois	18
Or	22

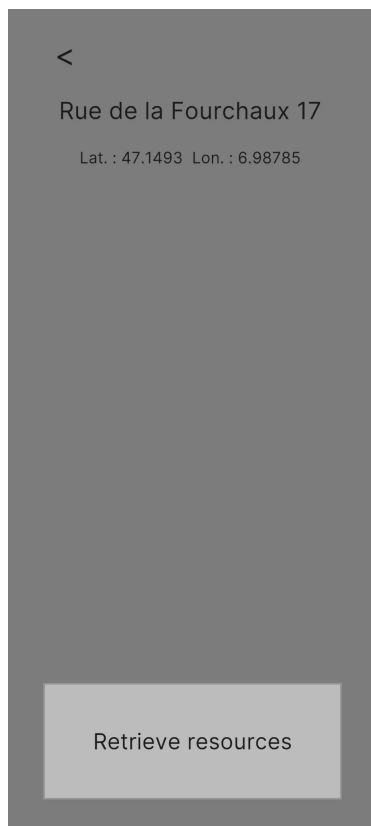


FIGURE 5.10 – Interaction avec un point d'intérêt

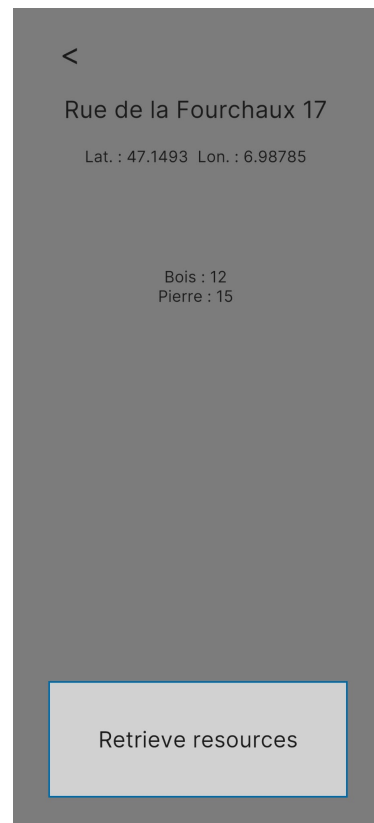


FIGURE 5.11 – Interaction avec un point d'intérêt - Génération de ressources

5.3.4 Château

FIGURE 5.12 – Menu du château



5.3.5 Erreurs bénignes encore présentes

Bien que l'application soit fonctionnelle et stable à l'état actuel. Il reste deux erreurs bénignes étranges qui n'ont pas pu être corrigées jusqu'à ce jour.

La première est une erreur visible qu'en mode "Éditeur" de unity. Quelques fois, et pas tout le temps, lors de la connexion une erreur indiquant l'utilisation d'un conteneur primaire génère une fuite de mémoire. Seulement, l'erreur ne donne aucune ligne de code et la source est introuvable. Cela provient probablement d'un objet qui est mal détruit.

La seconde est une erreur liée au SDK de Google Maps, lorsque le point central de la carte est initialisé une nouvelle fois alors qu'il l'est déjà, une erreur surgit et indique que ça n'est pas possible. Cependant lorsque l'on ajoute une condition pour éviter cette erreur, le système ne fonctionne plus du tout. Comme l'erreur ne fait rien et ne stoppe pas le jeu, il a été décidé de la laisser comme telle.

Chapitre 6

Conclusion

Le but du projet était de créer un jeu possédant les caractéristiques et les fonctionnalités du SDK de Google Maps et ce dans son entièreté. Il était nécessaire de récupérer le projet d'un ancien étudiant du nom de M. Dos Santos Ferreira Julien et de l'adapter en ajoutant des éléments d'un jeu inventé à destination d'Android.

Comme cité plus haut dans le chapitre Résultats, Camelote est dans un état stable et possède plusieurs possibilités d'améliorations intéressantes.

En conclusion, le projet P3 Camelote se termine en respectant la quasi-totalité des objectifs principaux du cahier des charges. L'application offre un système de connexion et un moyen de s'inscrire et une fois connecté, l'utilisateur a accès au jeu. Il peut alors récupérer des ressources en interagissant avec les points d'intérêts aux alentours et les utiliser pour améliorer son château.

Bien que les objectifs secondaires et tertiaires n'aient pas pu être respectés, le jeu se trouve dans un état complet et stable.

Bibliographie

- [1] B. to Bits Games, “Unity + MySQL playlist.” [Online]. Available : https://www.youtube.com/watch?v=CO_DK75XO14&list=PL5KbKbJ6Gf99mcmE1ptsn0oXO1_vnKDI&ab_channel=BoardToBitsGames
- [2] phpdelusions, “SELECT query with variables.” [Online]. Available : https://phpdelusions.net/mysql_examples/select
- [3] —, “UPDATE query from an array.” [Online]. Available : https://phpdelusions.net/mysql_examples/update
- [4] —, “INSERT query from an array.” [Online]. Available : https://phpdelusions.net/mysql_examples/insert
- [5] D. Dino, “How To Use Unity’s New UI BUILDER.” [Online]. Available : https://www.youtube.com/watch?v=EVXFWXaZtjw&ab_channel=DapperDino
- [6] —, “How To Create Dynamic UI With The NEW UI Toolkit - Unity Tutorial.” [Online]. Available : https://www.youtube.com/watch?v=6zR3uvLVzc4&t=1s&ab_channel=DapperDino
- [7] C. Code. [Online]. Available : https://www.youtube.com/watch?v=NQYHIH0BJbs&t=578s&ab_channel=CocoCode
- [8] LanKuDot. [Online]. Available : <https://forum.unity.com/threads/ui-builder-doesnt-save-the-reference-to-another-uxml-when-saving-twice.993007/>
- [9] Unity-Technologies. [Online]. Available : <https://github.com/Unity-Technologies/UIToolkitUnityRoyaleRuntimeDemo>
- [10] D. Tutorial. [Online]. Available : <https://dots-tutorial.moetsi.com/ui-builder-and-ui-toolkit/create-a-list-view-in-unity-ui-builder>

Table des figures

2.1	Maquette d'un menu principal	5
2.2	Maquette de l'interface du jeu	5
2.3	Maquette du menu de jeu	6
2.4	Maquette du menu des ressources	6
2.5	Maquette du menu des unités	6
2.6	Maquette du menu du château	7
3.1	Diagramme de séquence - Interaction avec un point d'intérêt	10
3.2	Diagramme de séquence - Montée de niveau d'une unité	10
3.3	Diagramme de séquence - Entrée dans une arène	11
3.4	Schéma de base de données	12
3.5	Diagramme de package	13
4.1	Récupération des points d'intérêts	26
4.2	Le visuel dans l'éditeur	27
4.3	Le visuel sur le téléphone	27
5.1	Menu principal du jeu	30
5.2	Menu de connexion	31
5.3	Erreur dans le menu de connexion	31
5.4	Menu d'inscription	31
5.5	Inscription de l'utilisateur correcte	31
5.6	Erreur dans l'inscription de l'utilisateur	31
5.7	Interface de jeu	32
5.8	Menu du jeu ouvert	32
5.9	Menu des ressources	32
5.10	Interaction avec un point d'intérêt	33
5.11	Interaction avec un point d'intérêt - Génération de ressources	33
5.12	Menu du château	33

Annexes

- [1] Cahier des charges - Dans le dossier 010-RapportFinal-Annexes
- [2] Diagramme de GANTT - Dans le dossier 010-RapportFinal-Annexes
- [3] Page Gitlab du projet Camelote - <https://gitlab-etu.ing.he-arc.ch/isc/2021-22/niveau-3/projets-p3-dlm/208>
- [4] Journal de travail - Dans le dossier 010-RapportFinal-Annexes
- [5] Documentation du projet "Guided Tour Neuchatel" - Dans le dossier 010-RapportFinal-Annexes